
Physics-Based Real-Time Sound Synthesis for Virtual Reality Musical Instruments (VRMI)

State of the Art, Design and Implementation of VRMIs

Master Thesis
Lui Albæk Thomsen

Aalborg University Copenhagen
Department of Architecture, Design & Media Technology
A. C. Meyers Vænge 15
2450 Copenhagen SV



AALBORG UNIVERSITY
STUDENT REPORT

**Department of Architecture, Design &
Media Technology**
A. C. Meyers Vænge 15
2450 Copenhagen SV

Title:

Physics-Based Real-Time Sound Synthesis for Virtual Reality Musical Instruments

Theme:

Master Thesis

Project Period:

Spring Semester 2017

Participant(s):

Lui Albæk Thomsen

Supervisor(s):

Stefania Serafin

Copies: 1

Page Numbers: 60

Date of Completion:

August 15, 2017

Abstract:

With the recent wave of development in Virtual Reality (VR) technology, new purposes have been found for the medium. The complex motion tracking and limitless environments allows for interesting new ways of interacting with musical instruments. Recently, researchers have examined the concept of musical interaction in VR, which has led us to a new category in expressive musical interfaces; Virtual Reality Musical Instruments (VRMI). The thesis project presents interactive prototypes with integrated physics-based sound synthesis through an iterative design process of usability and crossmodal association evaluation. The main conclusion of the usability evaluation includes further improvements to the robustness between gesture mapping and collision detection system. From the crossmodal association experiments, a slight tendency towards successful identification of size and material of the sound producing object was found, however, given the small sample size and potential flaws in the experimental design, future work and evaluation remains a requirement for raising the validity of the indicative tendencies.

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Contents

Preface	vii
1 Introduction	1
1.0.1 Background	1
1.0.2 Summary of Research	4
2 Analysis	5
2.1 Mixed Reality	5
2.2 Music in Mixed Reality	7
2.2.1 Research	7
2.3 Sound Synthesis	9
2.3.1 FAUST (Functional Audio Stream)	9
2.3.2 FAUST-STK	10
2.3.3 Physical Modeling Synthesis	10
2.3.4 Digital Waveguide Synthesis	11
2.3.5 Finite Element Analysis (FEA)	11
3 Methodology	13
3.0.1 Virtual Reality	13
3.0.2 Crossmodal Associations	14
4 Design	15
4.0.1 VRMI Interaction	16
4.0.2 Non-Immersive Interaction	17
4.0.3 Summary of Requirements	17
5 Implementation	19
5.0.1 Plugin Workflow	19
5.0.2 Physical Model Parameters	20
5.0.3 Virtual Reality	21
5.0.4 Parametric Mapping using C# Marshalling	24
5.0.5 Revisited Version for Crossmodal Experiment	25

6	Evaluation	31
6.0.1	Usability Testing	31
6.0.2	Crossmodal Association	32
7	Discussion	35
7.0.1	Interaction Prototype (VR)	35
7.0.2	Crossmodal Experiment	37
7.0.3	Future Perspective	37
8	Conclusion	39
	Bibliography	41
A	Acronyms	43
B	Scripts	45
C	Virtual Reality Technology	51
D	Musical VR Applications	53
E	Physical Model Parameters	59

Preface

Thank you to Stefania Serafin for all the guidance, support and supervision of my thesis project and for most of my projects during my time as a student on the master's degree of Sound & Music Computing at Aalborg University Copenhagen. Another thank you to Smilen Dimitrov, who has been an excellent help as technical supervisor for many projects during my studying.

I would also like to take the opportunity to reach out with a special thank you to the developers of FAUST for the guidance and support during the development of the thesis project. Therefore, a thank you to Romain Michon from CCRMA at Stanford University for introducing me to the programming language and for getting me in contact with some of the very friendly and helpful developers of FAUST; thank you to Yann Orlarey and Stéphane Letz from GRAME (Centre National de Création Musicale).

Aalborg University, August 15, 2017

Lui A. Thomsen

Lui Albæk Thomsen
<lath11@student.aau.dk>

Chapter 1

Introduction

With the resurgence and commercialization of Virtual Reality (VR) technology, a new wave of interactive and creatively expressive musical interfaces is surfacing the community and world of VR. The interfaces have been investigated and coined as Virtual Reality Musical Instruments (VRMIs) [1]. Following the definition, a set of conceptual design principles has been established [1] with respect to prior research in the field of New Interfaces for Musical Expression (NIME). By default, VR is dependent on the same development platforms used in traditional game design for 3D experiences (i.e., game engines).

In terms of the audio, the wave of VR has pushed the abilities of game engines to spatialise audio in real-time using various newly developed libraries and plugins. However, it seems that the common approach (given e.g., Unity's features in terms of playback of sounds) for sound design is to use third-party software like Digital Audio Workstations (DAW) for synthesizing, processing and recording the intended sound design of a game.

In regards to musical experiences in VR, and the sophisticated and low-cost motion tracking that most common VR systems include, the idea of using motion data for interacting with musical tones generated by mathematics should be more accessible than ever. The following thesis project will pursue the task of investigating options for real-time embedment of sound synthesis in game engines intended for musical interaction in Virtual Environments (VE), while establish and discussing the technicalities in the development of interactive audio plugins for use in a VE. The thesis project will also document several implementations of real-time sound synthesis related to interaction with and realism of musical instruments.

1.0.1 Background

The following sections will briefly describe the core areas of research in the thesis project.

Virtual Reality The current state of the art VR technology allows for multiple tracking points (e.g., head and hands obtained from the Head-Mounted Display (HMD) and hand-held controllers) with positional and rotational information, which allows for complex gestures in controlling behaviour within games and VEs. The hand-held controllers enable the interfaces to be partly tangible with its physical form and interactive buttons, however, the actual virtual environment is intangible and absent of true haptic feedback – unless information about user actions is conveyed through the actuators of the hand-held controller.

The author of the thesis has previously co-authored a paper considering the use of VR for musical instruments [2]. Part of the paper concerned an implementation of virtual percussive instruments designed for a commercial VR system containing a HMD, hand-held controllers, and room-scale tracking (6DoF). The main conclusion from the evaluation of the design and implementation was that a lack of dynamic audio, related to the gestures being generated in real-time, decreased the playability of the VRMIs. The term of *playability* has been discussed in [3], however, throughout the following thesis project the term will be simplified to merely describe interaction as feasible for simple musical performances. A larger scale study on playability might be performed in the future covering all aspects of *playability*.

Synthesis Engines Various development platforms and programming languages like SuperCollider ¹, JUCE ², Max/PD ³, CSound ⁴ and FAUST ⁵ allows for implementation and control of real-time audio synthesis models and more. A strong motivator for the thesis project has been the intrusiveness of providing high-quality sound synthesis in VR experiences. In previous personal experiences (as well as other research projects), the audio engine has been interoperable through an Open Sound Control (OSC) ⁶ network protocol connection with the aforementioned third-party software. However, such an approach requires additional software to run on the side, as well as an implemented library for setting up the server and listener of the protocol. As VR is usually highly dependent on visual content, interactive virtual environments are designed similarly to traditional video games played on a screen. Therefore, the embedment of the *sound synthesis engine* into the executable VR experience is interesting for the prevalence of musical instruments in VR that are convenient and easy to use.

Apart from avoiding intrusive software setups, the thesis project will also be developed under the same principle in relation to the involved hardware. As described previously, the current state of the art in VR technology has certain restrictions on the types of inputs that can be provided to the system, and is mostly based on tracking of motions.

¹SuperCollider: <http://supercollider.github.io/>

²JUCE: <https://www.juce.com/>

³Max: <https://cycling74.com/products/max>

⁴CSound: <http://csound.github.io/>

⁵FAUST: <http://faust.grame.fr/>

⁶Open Sound Control: <http://opensoundcontrol.org/>

By understanding the limitations of modern VR technology, the appropriate musical instruments can be chosen for implementation in a non-intrusive software and hardware approach. By studying the field of organology, the physical realms involved in the production of sound by musical instruments can be explained. Developing realistic VRMIs requires the multisensory experience of VR to be considered. In VR, there is a close relationship between what is seen, heard and felt. With traditional Virtual Musical Instruments (VMI), the interaction might be physically present in the real-world through a simplified interface controlling physical models, however, in VR, the physical model can become part of the simulated reality.

Musical Instruments Classification of modern musical instruments goes back to the beginning of the last century, when German and Austrian musicologists Curt Sachs and Erich von Hornbostel [4] organized musical instruments into various distinctive classes. The distinction mainly considered the vibrating element of the musical instruments; bodies, membranes, strings or columns of air. Thereby, the categories were named idiophones (body), membranophones (membrane), chordophones (strings), and aerophones (columns of air). The classification was reevaluated by André Schaeffner in 1932 [5], expanding the classification for any conceivable instrument. The new classification organised idiophones, membranophones, and chordophones into the collective of gaiaphones, and maintained aerophones as its own category. Thereby, the categorization was now based on the physical organology of the musical instruments with respect to the bonding of atoms responsible of producing sound. The table below provides an overview of the involved forces and gestures with musical instruments.

Name	Gaiaphones	Hydraulophones	Aerophones	Plasmaphones	Quintephones
State	Solid	Liquid	Gas	Plasma	Quintessence
Bonding	Solid bonds	Weak bonds	No bonds	Ionization	Undefined
Material	Vibrating body, membrane, string	Vibration in medium	Vibrating columns of air	Atoms	Electricity, amplification, processing units
Examples	Idiophones, membranophones, chordophones	Reed-based, reedless	Woodwind instruments, Brass instruments	Pyrophone, Rijke tube, Tesla coil	Mechanophones, electrophones, optiphones, neural networks
Gesture	Striking, bowing	Complex	Blowing	Complex	Complex

Table 1.1: Organological classification

By knowing the involved physics of musical instruments, the area of simulation in real-time games and VR experiences can be investigated. The main limitation to start off with is the support of VR development. Currently, the two widest used game engines for designing VR experiences are the Unreal ⁷ and Unity ⁸ game engines. Both game engines make use of NVIDIA’s PhysX ⁹ physics engine, which includes features such as gravitation, collision detection, rigid and soft body dynamics, particles and more. It might be that we in the future are able to simulate sound by the same means of rigid body animations, however, at the current moment, it might be more feasible to investigate the possibilities in prebuilding models based on physics instead of entirely integrating the simulation process with the physical laws of the game engine.

⁷Unreal Engine: <https://www.unrealengine.com/>

⁸Unity: <https://unity3d.com/>

⁹NVIDIA PhysX: <https://www.geforce.com/hardware/technology/physx>

Crossmodal Association The ability of the synthesis engine to induce cross-modal associations between visuals and sound components is also an important factor in the evaluation of the implementation of the thesis project. The crossmodal correspondence between frequency and spatial properties of a sound was investigated in [6] and provides insight in the state of theory on crossmodal correspondence between pitch and sound objects. For example, it was shown in 1984 by Walker and Smith that high-pitched sounds and small objects share crossmodal qualities and vice versa in relation to larger objects and low-pitch sounds. The phenomenon was concluded to be explained by the perception of resonance in objects. Inanimate objects resonate in an inverse manner depending on their frequencies (larger objects tend to resonate at lower frequencies and vice versa), while gender and size also creates associations with pitch (e.g., males tend to have a deeper voice). Moreover, the study conducted a test on the correspondence between frequency (Hz) and the diameter of a two-dimensional circle. 21 tones were presented to test participants and they were asked to match the sound with 9 circles varying in diameter. Conclusively, the study showed a strong association between pitch and size, which is consistent with previous research [6].

1.0.2 Summary of Research

In summary, the topics of VRMI, multimodal VR systems, previous research conclusions, real-time sound synthesis, software nativity, physics and classification of musical instruments, physics simulation in game engines, and crossmodal associations has been briefly covered in the current section providing background knowledge in the field. The main point for investigation to be taken from the introductory sections is the feasibility of implementing real-time sound synthesis for realistic and interactive VRMIs within the development environment of a game engine. The research areas prone to analysis will be Mixed Reality (MR), the development options for compiling a plugin for a given game engine, the mathematics behind simulation of musical instruments based on physical modeling, and lastly the state of theory in VRMI research.

More specifically, the following chapters will investigate reality-virtuality continuum, design principles for developing VRMIs, physical modelling in FAUST, and plugin compilation in C++ for the Unity Audio Engine.

Since the publication of the theorem, the quality and affordability of compact digital displays have greatly increased and products enabling the concepts described in the continuum are now commercially available.

At the current moment, the VR market is slightly saturated by many innovation start-ups and established technology manufacturers. Generally, the industry has not yet settled on common approaches and solutions for output devices, which means that many systems are marketed as peripheral equipment for VR. In 2016, a conference paper on the State of the Art in VR technologies was published [7]. The paper provides a good overview of the current state of VR hardware. The paper summarizes input and output into the categories of visual, haptic, and multi-sensory input/output devices and controller, navigation, and tracking input devices. The classification is relevant for the aforementioned classification of musical instruments to provide an overview of the possibilities within the market between VR systems and physical sound models. A hierarchical graph showing specific products related to the categories can be found in Appendix C.

On the software side of VR, there are multiple attempts at creating a standardized way of implementing interaction and, most recently, cross-compiling of VR applications to any VR hardware. In regards to the implementation of interaction, the two toolkits of NewtonVR ¹ and VRTK (Virtual Reality ToolKit) ² both already have many commercial VR titles using their interaction implementations. In regards to cross-compiling of VR applications, the need for a VR development standard was identified and the Khronos Group has taken the initiative (together with companies such as AMD, Google, Intel, Nvidia, and Samsung) to create a VR standard named OpenXR ³. The vision of the Khronos Group is outlined in the image below.

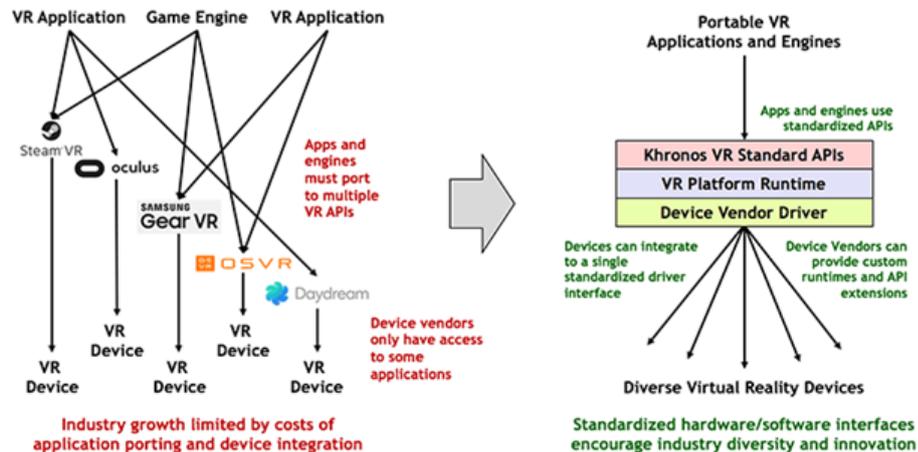


Figure 2.1: Khronos VR

¹NewtonVR: <http://www.newtonvr.com/>

²VRTK: <https://vrtoolkit.readme.io/>

³OpenXR: <https://www.khronos.org/openxr>

2.2 Music in Mixed Reality

The following section will provide an overview of the current state of research in musical interaction for MR applications.

2.2.1 Research

With the recent focus by hardware manufacturers on development of VR hardware, the field of musical instruments in VR is quite new and yet to be fully explored. However, in 2016, a new paper by researchers from Aalborg University Copenhagen [1] has been published with a set of established principles for designing Virtual Reality Musical Instruments (VRMI). The paper investigates the state of the art in previously described virtual instruments. However, as mentioned, the keyword of virtual has been used to describe software simulated musical instruments visualized by CGI. By the new possibilities in creating “actual” virtual environments through binocular vision and motion tracking, the paper also recategorizes the previous virtual attempts by distinguishing between Digital Musical Instruments (DMI) and Virtual Reality Musical Instruments (VRMI).

The paper also proposes a set of nine design principles for developing VRMIs. The design principles are outlined below with a summarized description of each[1].

- Design for Feedback and Mapping
 - Emphasizes the importance of designing multimodal feedback for the interaction between the user and the instrument.
- Reduce latency
 - Emphasizes the importance of synchronicity between sensory feedback.
- Prevent cybersickness
 - Conflicting sensory feedback (especially in visual and vestibular stimuli) should be avoided or kept at a minimum.
- Make use of Existing Skills
 - Emphasizes the importance of creative solutions for the interaction by avoiding replication of instruments from the real world.
- Consider both Natural and "Magical" Interaction
 - Make use of the unlimited possibilities in virtual interaction that is not constrained by the laws of physics or human anatomy.
- Consider Display Ergonomics
 - Be aware of the VR hardware industry still being in a prototypical stage, especially in regard to the need of a tethered connection between display and computer.
- Create a Sense of Presence

- Evaluate and design the experience with respect to previous research on presence in VR.
- Represent the Player’s Body
 - Emphasizes the importance of providing a sensation of virtual body-ownership in VR.
- Make the Experience Social
 - Emphasizes the potential positive impact of a social setting in VR on performance and the experience itself.

In regards to virtual body-ownership, another study related to rhythm and VR came out in 2012. The study focused on the appearance of a virtual body and the relation to the person playing the percussive instrument. The study showed that full body-ownership illusions has great impact on behaviour and performance, moreover, possibly also influencing cognitive effects of the sensation of virtual body-ownership [8].



Figure 2.2: Virtual Body-Ownership

Musical tutoring using VR has also been investigated in the project of *VRMin* [9]. A mobile VR application was developed and evaluated using a heuristic evaluation approach together with performance logging. The visual cues of the project consisted in informing the performer about hand-position and relation to notes and amplitude.

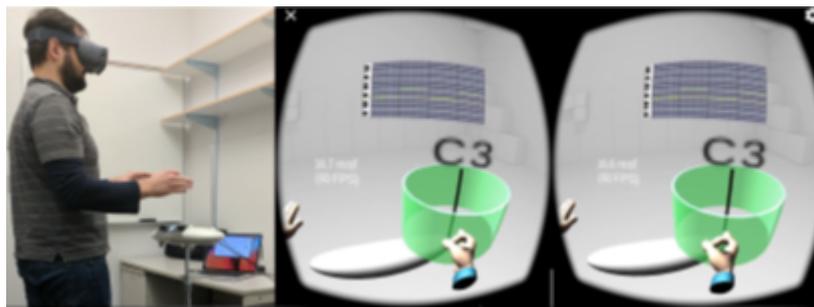


Figure 2.3: VRMin in action

The Theremin (the “replicated” musical instrument of the project) is notoriously known to be an instrument that is hard to pick up and learn due to its reliance on

contactless performance. An interesting outcome of the project was a set of design guidelines for heuristic evaluation of VEs intended for musical tutoring [9].

- Visual feedback should not prevent a user from focusing on aural feedback
- Reduce cognitive load by limiting the amount of concurrent feedback
- Provide terminal performance feedback upon completion of the practice task

These three guidelines should aid in the design phase to ensure that focus is not taken away from the actual task of a VRMI experience; musical performance. The project also considered the design principles presented by Serafin et al., which was covered previously in this section. Conclusively, the project's evaluation resulted in a set of design modifications for the future perspective of the project, in which a larger scale user study will further investigate the influence of various environmental configurations.

An overview of the current state of the art in commercial entertainment solution related to music and rhythm in VR can be found in Appendix D.

2.3 Sound Synthesis

The following sections will elaborate upon the ways of synthesising sounds in real-time with the programming language of FAUST. An explanation on the physical modelling involved will also be provided.

2.3.1 FAUST (Functional Audio Stream)

FAUST is a functional programming language designed for real-time signal processing and synthesis. It consists in a compiler that translates a FAUST program into an equivalent C++ program optimized for efficiency. The compiler includes many options for building the C++ program into various architectures. Thereby, the compiled FAUST program can be either specifically aimed at architectures like PD, Android, iOS, etc. or implemented in other environments that support C++ programs. The supported plugin architectures are LADSPA plugins, CSOUND opcodes (incl. double precision), Max/MSP externals, Native/Windows VST plugins, Supercollider plugins, Puredata externals, Q plugins, Pure plugins. Moreover, FAUST also supports the use of the networking protocol OSC for interoperability across software.

Compatibility

FAUST is a specification language in which signal processors can be programming using the semantically driven compiler. The compiler translates the mathematical functions into efficient C++ (and C, Java, Javascript, LLVM IR, and WebAssembly) programs. By that principle, a number of C++ platforms are supported using the `faust2api` commands. Recently, compiling options for the game engine Unity was added to the distribution, which natively supports C++ plugins for extending the

Table 2.2: Process

PhysicalParameters → *PhysicalModel* → *OutputSound*

functionalities in real-time. A program can thereby be designed Faust, compiled with instructions for the Unity Audio Engine and through C# marshalling be dynamically changed at run-time.

2.3.2 FAUST-STK

A set of virtual musical instruments based on physical modelling is part of the Faust distribution. The work was presented in [10] and includes waveguide models for wind (e.g., clarinet, flute, brass), string (e.g., bass, bowed string, sitar), and percussion (e.g., Tibetan bowl, iron bare, glass bare) instruments. A full list of the instruments can be found in the footnote ⁴. The implementations are based on the open source API Synthesis Toolkit developed and maintained by Perry Cook (Princeton University) and Gary Scavone (McGill University) [11]. The code was slightly changed to adapt the models to the semantics of FAUST. As described in [10], wind instruments are based on breadth pressure that corresponds to the amplitude of the excitation is controlled by an envelope. The excitation is used to feed one or several waveguides that implement the body of the instrument.

Part of the implementation for string instruments was already accessible through the processing libraries within FAUST of `filter.lib` and `effect.lib`. The models were spectrally enriched using non-linear all-pass filtering. The non-linearity of the filters is generated by dynamically modulating the filter coefficients at every sample by a function of the incoming signal [10].

Percussion instruments of FAUST-STK use excitation by a bow or a hammer and are based on banded waveguide synthesis. Banded waveguide synthesis simulates acoustic dispersion of an object - a wave is influenced by a medium or material and separates its component frequencies - or inharmonic resonant frequencies of an object [10]. The following sections will describe physical modelling and waveguide synthesis for musical instruments.

2.3.3 Physical Modeling Synthesis

In physical modelling, a desired waveform is generated by an established model consisting of mathematical equations and algorithms. The model is oftentimes constructed to emulate the physical properties of, for example, a musical instrument. The physical properties (or parameters) are related to the instrument's constructional dimensions, materials, and mechanics that all influences the final sound. The parameters can also be defined by the performer's interaction with the instrument [12].

⁴FAUST-STK overview: <https://ccrma.stanford.edu/~rmichon/faustSTK/>

In music, there are generally two distinctive models in use for sound synthesis: lumped or distributed. Lumped models describe masses, springs, dampers, and non-linear elements, whereas distributed models describe wave propagation in strings, bores, horns, plates, and acoustic spaces. Lumped models are typically implemented by a two-pole digital filter and distributed models consist of a combination of digital waveguides, digital filters, and non-linear elements [12].

2.3.4 Digital Waveguide Synthesis

Musical instruments can be computationally modelled using delay lines, digital filters, and non-linear elements. Such a signal processing chain is referred to as a *digital waveguide synthesis model* [12]. Between each model there are the following similarities:

- Sampled acoustic traveling waves
- Follow geometry and physical properties of a desired acoustic system
- Efficient for nearly lossless distributed wave media (strings, tubes, rods, membranes, plates, vocal tract, etc.)
- Losses and dispersion are consolidated at sparse points along each waveguide.

Traveling waves can be simulated by a bidirectional delay line at wave impedance R , which outputs a physical signal [12].

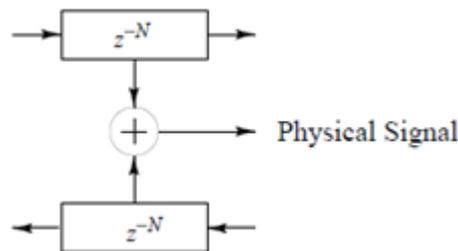


Figure 2.4: Output Signal

2.3.5 Finite Element Analysis (FEA)

A Finite Element Analysis (FEA) (also known as the Finite Element Method (FEM)) is a numerical method for analysis of structures, solid mechanics, dynamics and more. FEA provides an approximate solution to the task of calculating displacement, stress and strains at various points of a material when force is applied. The concept is illustrated below with material points X and force P [13].

As understood from the conceptual illustration, the undeformed solid state hold a number of measurable quantities related to the mechanical physical property of the object. A large number of physical properties can be measured and provided as input

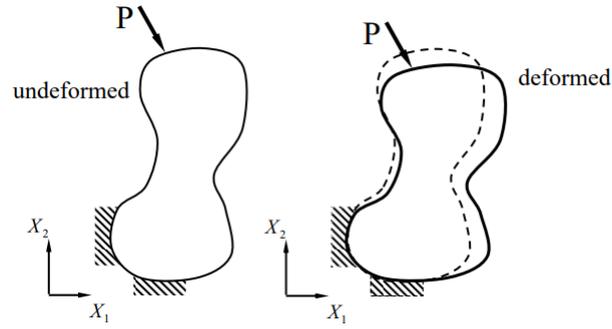


Figure 2.5: Force applied to solid resulting in deformation [13]

to implementations of FEA [13]. Basic properties related to the boundary (e.g., a 3D mesh) and the solid interior structure (e.g., density, elasticity) are commonly used as parameters for FEA algorithms. Basic physical properties of materials include the measures of Young's modulus, Poisson's ratio and density.

Young's modulus is a unit used to describe the rigidity of an object of solid material. In other words, it tells us about a materials ability to deform while being exposed to a force. The unit of Young's modulus is pascal due to its relation to pressure. The equation can be found below:

$$E = \frac{\textit{Stress}}{\textit{Strain}}$$

Poisson's ratio is the negative ratio between a longitudinal and lateral strain being applied to material. Poisson's ratio is related to the phenomenon of Poisson's effect, which describes a materials ability to compress or stretch when longitudinal force is applied. Poisson's ratio does not have a specific unity, but is often described by the μ symbol as seen in the equation below:

$$\mu = -\frac{\textit{Lateral strain}}{\textit{Longitudinal strain}}$$

Finally, the property of density is a substance or material's mass per unit volume. The SI unit of a material's density is $\frac{\textit{kg}}{\textit{m}^3}$ and is calculated by the following equation:

$$\rho = \frac{m}{V}$$

Chapter 3

Methodology

The evaluation of the design and implementation will focus on usability testing (playability in the context of music) and crossmodal matching tasks. The design and implementation of various prototypes will both involve traditional game play interaction with a keyboard on a 2D screen, as well as with motion tracked hand-held controllers and HMD.

3.0.1 Virtual Reality

The evaluation of VR applications will be performed with the user experience of the application in focus. The related researchers and methodologies will be presented below.

In correspondence to the evaluation methods of the state of theory in VR, usability and user experience can be tested according to evaluation methods presented in [14] [15] [16] [17].

The methodologies presented in the publications on presence and user experience can be used post-sessionally to evaluate the most recent experience. Qualitative data will be obtained verbally through interviews or in written questionnaires. Analysis of the qualitative data can be performed by the inclusion of self-reported assessment ranking in the form of Likert-scales. Quantitative data can be obtained by the implementation of data logging while using the MR applications. Both the qualitative and quantitative data will be used for the process of reiterating the design of the MR applications. Once mistakes are identified, optimization or rejection of core and new features can be performed.

The exemplary implementation in the thesis project will be evaluated from heuristic approach with the usability (i.e. playability for musical instruments) in focus. Features and design choices will individually be questioned through the observations during interaction and evaluated on test participants using questions from the standardized usability testing like the UEQ questionnaire. Throughout the development process, pilot tests will be performed to evaluate certain approaches, minor changes, and interaction threshold constraints. The factors in a successful design will be based

on the design principles and guidelines found in the previous section of Evaluation. Findings from the evaluation will be presented through commentary transcripts of user feedback from the testing sessions.

3.0.2 Crossmodal Associations

The evaluation and data acquisition of the crossmodal associations between visual and auditory components will be performed by a match-to-sample task between sensory stimuli. The methodology follows the evaluation methods of aforementioned studies related to crossmodal perception [6]. Test participants will be presented with an auditory stimulus and asked to choose the associated visual component related to the sound. A questionnaire will be used to obtain the reportings of crossmodal associations.

Chapter 4

Design

In the previous chapter and sections of 2 and 1.0.1, the topics of physical modelling, organology of musical instruments, and FEA was described. The exposition on the classification of musical instruments based on physics will now be used together with physical models to provide an overview of what is possible within the scenario of musical interaction in VR. Apart from the creative approach of natural or magical musical interaction, the physical models can also be investigated in terms of realism and compellingness in the relationship between audio and visuals. Such a design does not necessarily require implementation in a VE, as important factor here is the crossmodal correspondance perceived by static users.

Delimitation As presented in 1.0.1, there can be said to exist five ways that musical instruments produce sound. The required input modality can be used in the context of sensors enabling a system to simulate the same sensory interaction. It should be noted that interaction with musical instruments can semantically be done through protocols (i.e., MIDI or OSC) connecting the sound synthesis to any digital interface. The digital representation can come in many forms and is commonly used by keyboards with one or multiple playable octaves. However, given the multimodality of VR and its close relation to actual reality, the requirements should accommodate true interaction with said musical instruments. Recalling the classification of musical instruments: gaiaphone, hydraulophones, plasmaphones, and quintepphones.

Due to the complexity of materials involved in hydraulophones, plasmaphones and quintepphones, an implementation would require extensive physics simulation of liquid and electricity. Currently, these musical instruments will be excluded from the requirements due to the abilities of game engines, which ultimately decides the simulations unless complex extensions are added to the current state of the art in VR technology. However, it should be noted that the category of quintepphones is arguably the most fitting for any musical instrument implemented for VR interaction given its abstract and unlimited concept.

4.0.1 VRMI Interaction

With the exclusion, the remaining categories of gaiaphones and aerophones can be discussed in a VR setting. The discussion will be based on the VRMI design principle of *Natural/Magical interaction* (see Section 2.2) and provide potential solutions for both interaction types. Properties of the musical instruments is summarized in the table below.

Types	Chordophones	Idiophones	Membranophones	Aerophones
Example				
Vibrating element	String	Body	Membrane	Air
Gestural interaction	Striking	Striking	Striking	Blowing
Digital detection	Collision	Collision	Collision	Air pressure change
Sensor type	Optical/Inertia	Optical/Inertia	Optical/Inertia	Anemometer/Microphone

Table 4.1: Gaiaphones and Aerophone Classification

By deriving the type of interaction and required technological sensors from the vibrating elements of the musical instruments, a set of requirements can be formed for a VR system enabling interaction with said instruments. From analysis of the state of the art in VR technology [7], it is known that no systems currently include sensor types for aerophones (i.e., anemometer or microphone). However, a microphone would be possible to integrate, but become intrusive given the attention hardware required.

It should be noted that the entire discussion here is based on the principle of implementing a system that allows for natural interaction with the musical instruments. However, in the music production industry, digital interfaces (e.g., MIDI keyboards) are commonly used to interact with any given synthesis module that is compatible with the standardized MIDI specification (i.e., Note on/off, key number, velocity, etc.).

In regards to chordophones, idiophones, and membranophones, the implementation is possible through prebuilt physics-based models or potentially the physics engine of a game engine. As described in 1.0.1, common game engines includes the PhysX physics engine, which can deal with collision detection between virtual objects in real-time. This allows for the gestural interaction of striking, which is naturally done with the aforementioned musical instruments. From a hardware perspective, it is also known from the analysis presented in [7] that most VR systems include both inertial sensors (embedded in either the HMD or hand-held controllers) and optical tracking solutions for room-scale interaction. This will allow for real-time detection of collision between real-world motions and virtual objects to trigger musical events.

A set of requirements can be established for designing interaction with VRMIs. As part of the requirements, the solutions for of mapping gestures to the sound synthesis model are also proposed. The requirements are summed up in the bullet points below.

4.0.2 Non-Immersive Interaction

In the case of designing non-immersive (e.g., the absence of a HMD) interaction with the synthesis of a physical model, the user is restricted to keyboards with notes mapped to various layouts. The layout with a traditional computer keyboard requires mapping with physical model parameters and the buttons of the computer keyboard. Additionally, the mapping can be redirected in the syntax of MIDI for use with external hardware allowing for note control with a musical keyboard.

4.0.3 Summary of Requirements

The chapter of design will be summed up in various lists of requirements related to the interaction with physical models in a VE. The lists will serve as guidelines in the process of implementation to make sure that the design follows previous research and preliminary conclusions found in the thesis report. It should be noted that the term of immersion is merely used below to distinguish between media perceived on a two-dimensional display and through a head-mounted display with stereoscopic vision.

- Immersive Interaction
 - Natural interaction in current VR systems can be implemented for chordophones, idiophones, membranophones by collision detection
 - "Magical" interaction in current VR systems chordophones, idiophones, membranophones, and aerophones by (e.g., MIDI) mapping
- Non-Immersive Interaction
 - Mapping of notes between physical model and keyboard buttons
 - Interface for parametric configuration
 - MIDI wrapper for mapping of notes and parameters with external MIDI compatible hardware
- Technical Requirements (VR)
 - High-definition HMD with inertial and optical tracking
 - Motion tracked hand-held controller with physical interface
 - Software Development Kit compatible with major game engines
 - Game development platform with physics simulation supporting collision detection
- Interoperability
 - Integrated in game engine using marshalling and mapping
 - Third-party software or hardware using Open Sound Control (OSC) or Musical Instrument Digital Interface (MIDI)

Chapter 5

Implementation

The following section will show and describe an exemplary implementation of musical expression in VR. The main features will be the following:

- Compatible with the HTC Vive VR system
- Implemented using the Unity game engine and SteamVR API
- Natively supported real-time physical modelling synthesis using FAUST
- Sound configurability through hand-held controllers or Graphical User Interface (GUI)

5.0.1 Plugin Workflow

The implementation of the project started by the compilation process of the Unity plugin. The process is visualised in the diagram below. The first three steps concern the compilation of the plugin, whereas the last three concern the mapping of parameters between sound synthesis and gestures.

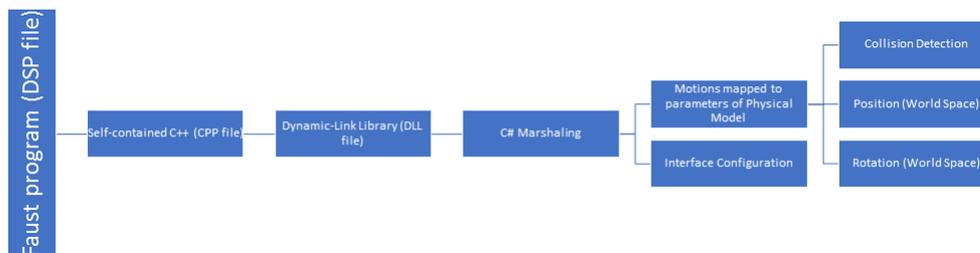


Figure 5.1: Workflow for Plugin Development

Unity supports native plugins written C++ that are compiled into a Dynamic-Link Library (DLL). Given that FAUST is written in C++, this seems like the optimal solution for building the plugin. The process starts by the .dsp file, which contains instructions of the program for digital signal processing and synthesis. The entirety of FAUST is only partly supported on Windows, however, the tools of

FaustLive and Faust Online Compiler provides similar compilation features as found in the main FAUST distribution. It is therefore possible on a Windows machine to upload a .dsp file and configure the compilation for the intended architecture and formatting. The FAUST Online Compiler (and FaustLive) has recently (spring 2017) added compiler options for Unity on iOS/OSX. However, the compiler still provides the option for a self-contained .cpp file, which can be modified to work on Windows as well. Windows is an important technical requirement to the implementation, due to VR systems exclusively working on high-end computers running Windows. Once the self-contained C++ program is obtained through the FAUST Online Compiler, the file can be added to a Visual Studio project together with a header file obtained from the online repository of Unity Audio Plugin SDK (<https://bitbucket.org/Unity-Technologies/nativeaudioplugins/src>). Three modifications were made to fix issues from compiling:

- A namespace must be declared around the DSP and synthesis processes of the program
- The name of the plugin must be defined
- Numbered labelling for the parameters instead of strings to avoid issues with prefixes

Once this was done, the last thing is to add the name of the plugin to the PluginList.h header file. Building the Visual Studio solution will generate the DLL-file contained all audio processes, parameters, and naming. By standard, Unity checks for the file name within the Assets folder and the default prefix for validation is to add “UnityAudioPlugin” before the relevant name of the generator or effect. Once this is done, the synthesis model can be added to a Unity Mixer Group and parameters are visible in the inspector of said object on a mixer channel. The further process of marshalling values and mapping parameters between the synthesis model and motions is discussed in Section 5.0.4.

5.0.2 Physical Model Parameters

The design and implementation of FAUST-STK allows for changing of parameters of the physical model in real-time through either a GUI or an OSC connection. The physical models include either partly or all of the following parameters:

- Basic Parameters (e.g., frequency, gain, note on/off)
- Physical Parameters (e.g., excitation selection, bow pressure/position, etc)
- Nonlinear Filter Parameters (e.g., modulation type/frequency, nonlinearity, etc)
- Envelopes (e.g, ADSR durations)
- Vibrato (e.g., frequency, gain, ADSR, etc)

A full list of the parameters of the specific models implemented in the project (prototype 1 and 2) can be found in Appendix E.

Due to an issue with the prefixes on the parameters' labeling, the C++ code was modified to label each parameters by numbers. As an example, the parameters of a Tuned Bar physical model viewed in the Unity Inspector can be seen in the image below.

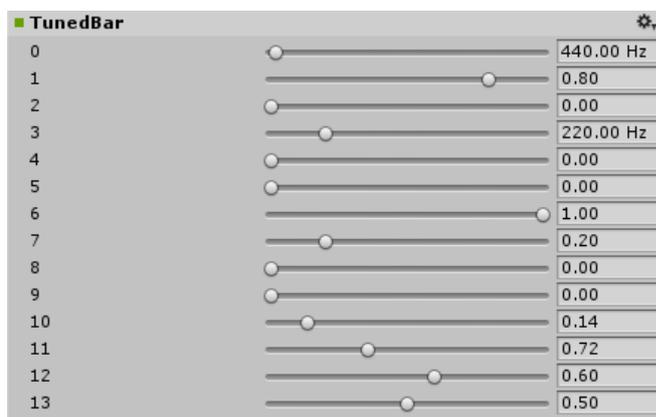


Figure 5.2: Inspector View of the Unity Audio Plugin

5.0.3 Virtual Reality

The following sections will explain and document an implementation of the interactive environment in VR using Unity and the HTC Vive. The HTC Vive is programmable using the SteamVR library obtained from the Unity Asset Store.

SteamVR

SteamVR can be imported as a custom package in Unity, however, room scale calibration and general setup of the HMD is done within Steam itself. By drawing the dimensions of the room with controller, the user defines the area in which it is possible to move around. The SteamVR runtime application is running at all times, and when Unity is set to render, the information regarding room dimensions is transferred to Unity – the accessible area is visually indicated by a blue boundary in the Unity editor. In VR, the boundary is shown when the user approaches the edge of the room by a distance threshold limit.

By the use of the SteamVR Camera rig prefab, the stereoscopic rendering is setup for the two eyes. In order to access and use the controls/buttons of the Vive Controllers, a script from the VRTK library was used to change inheritance properties of the objects. The main purpose of doing this was to enable the reaching and lifting of virtual objects by the controller.

Virtual Reality ToolKit (VRTK)

The Virtual Reality Toolkit is an asset collection of scripts and prefabs for rapid prototyping and designing of VR experiences. The asset is built for Unity and supports both the SteamVR and Oculus SDK. The collection of scripts and prefabs contains solutions for implementing locomotion, interaction gestures, UI elements, environmental physics in VR. The main use in the project has been to implement gestural interaction and configuration through buttons of the hand-held controller. The most essential scripts from the toolkit that was used in the implementation is that of the *VRTK_ControllerEvents* and *VRTK_BaseGrabAttach*. The scripts allows for manually configuring events related to the semantic actions of grabbing, holding, and releasing virtual objects. Respectively, the scripts enabled sound events to be mapped to the events of the controllers (i.e., spawning, translating, rotating and scaling a virtual object by the click of specific buttons on the hand-held controller). The *VRTK_ControllerEvents* is semantically built up in a way that allows for adding of specific events (e.g., transformations, collision detection, etc) within the defined events of *touch* (minimal button press), *press* (halfway button press), *click* (full button press). This event controlling is implemented for each of the buttons of *trigger* (7), *touchpad* (2), and *grip* (8), which are illustrated below.

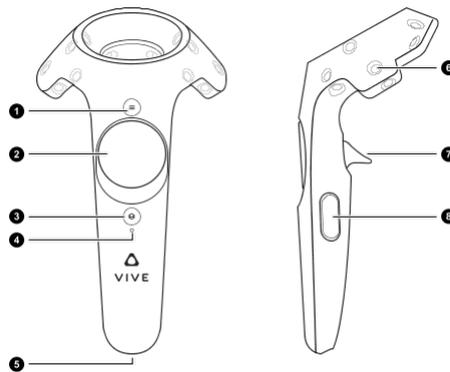


Figure 5.3: The hand-held HTC Vive Controller - 1. *Application Menu*, 2. *Touchpad*, 3. *System Menu*, 4. *Status Light*, 5. *Micro-USB Port*, 6. *Tracking Sensor*, 7. *Trigger Button*, 8. *Grip Button*

The implemented behaviour was mapped in the following way:

- **Trigger** spawns a prefab at the current position
- **Touchpad** changes position and rotation of nearest virtual object while pressed
- **Grip** changes scale of nearest virtual object by moving the secondary controller while primary controller's grip is pressed

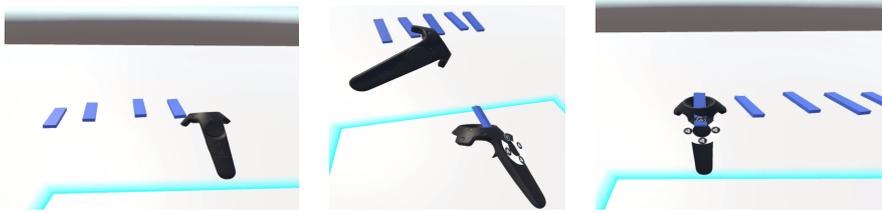


Table 5.1: Screenshots from implementation showing spawning, scaling, translating

Virtual Environment

The description of the virtual environment will be based on the content of the *Hierarchy View* in Unity. Conceptually, the environment consists of basic geometric primitives such as a plane for the floor, a rectangular box for the spawnable object, a cylinder for the interaction, and the more complex models of the HTC Vive controller, which appears during runtime and mapped to the position tracked by the system. By default, prefabs for SteamVR and VRTK has to be present in the Unity scene.

The *CameraRig* prefab from SteamVR handles the setup for stereoscopic vision, hand-held controller IDs, and room-scale tracking with the child objects of *Controller (left/right)* and *Camera (Head)*, where the latter contains setup for visual and auditory rendering through the camera and audio listener components.

The prefab of *VRTK* contains the SDK Manager and mapping options for the HTC Vive controller. The SDK Manager is used to define the system that the VR experience is designed for and contains options for SteamVR, Oculus VR, Daydream, and their own *Simulator* SDK, which can be used to design interaction without a VR system connected to the computer.

The two main objects enabling the interaction are the prefab *Bar* found in the assets folder and the *VRTK* child object of *RightController*. Conceptually, the controller setup enables spawning of the prefab by the click of the trigger button on the HTC Vive controller. The required scripts of the controller to enable such behaviour are the *VRTK_ControllerEvents*, which maps actions to certain buttons, and the script of *VRTK_ControllerEvents_Listener*, where behaviour is added in code for the previously defined actions. The main behaviour, which was added to the specific action of the trigger being pressed, is the instantiating of the prefab and is done by the following line of code:

```
Instantiate (bar, transform.position, Quaternion.LookRotation(transform.↵
right));
```

The parameters of the Unity function *Instantiate* are the game object, position and rotation that the object is intended to contain at spawning. The rotation has been implemented to follow the current rotation of the controller, ensuring consistency in the behaviour.

The first parameter of game object refers to the prefab *Bar*, which was premade to contain components of a collider, mesh renderer, the *VRTK_InteractableObject* script, *RigidBody*, the *VRTK_ChildOfControllerGrabAttach* script, the *VRTK_Axis-ScaleGrabAction* script, the customly made *Bar Tone* script, and an audio source. The VRTK scripts enable the object to be grabbed, transformed and scaled, whereas the *Bar Tone* is responsible for collision detection and mapping to the physical sound model. The collision detection was implemented using the function of *OnCollisionEnter* and checks for the tagged name of the cylinder used for striking the bars for triggering sounds. The mapping between motions and synthesis parameters is elaborated upon in the next section.

5.0.4 Parametric Mapping using C# Marshalling

In computer science, marshalling is the process of translating an object represented in the memory to a data format prone to be stored or transmitted between parts of software or programming languages. Marshalling was required in the project in order to enable the communication between the Dynamic-Link Library (DLL) compiled from C++ code to the scripting language of Unity, in which the game or experience is designed. The Unity runtime is written in C++, whereas the application's behaviour and UI is programmable in C#. While the implementation of marshalling on Windows can be done using the Marshal Class of the .NET framework ¹, Unity natively supports C# marshalling by exposing parameters viewed in the inspector of a native plugin. It is therefore possible to choose a parameter from the plugin, rename it for use in C coding, and changing the values by the function of *SetFloat*. The principle exists regardless of the implementation concerning animations, materials, or audio processes. In the case of the project, the purpose is to change parameters of the sound synthesis engine and thereby the appropriate way of dealing with this is to instantiate the Audio Mixer intended for playback and recalling the name of the parameter with a variable or static value that should be mapped. An example of mapping between physical model parameters and gestures can be seen below.

Physical Model Parameter	Transformation Parameter
Tone Frequency (Hz)	Y-position in World Space
Tone Gain (0 - 1)	X-rotation (Forward Vector)
Modulation Frequency (Hz)	Rotation Magnitude

Table 5.2: Parametric Mapping

Given the varying intervals of values, the range of each parameter needed to be remapped to obtain the effect of dynamic control. This task can be done by interpolation using a combination of the two mathematical functions of the Unity API; *Mathf.Lerp* and *Mathf.InverseLerp*. An example can be seen below with two different intervals and a variable (X) with the incoming data:

¹Marshal Class: <https://msdn.microsoft.com/en-us/library/system.runtime.interopservices.marshal.aspx>

```
float myVariable = Mathf.Lerp(low2, high2, Mathf.InverseLerp(low1, high1, X)←
);
```

5.0.5 Revisited Version for Crossmodal Experiment

An additional implementation was performed with the FAUST programming language. A subset of the distribution includes the tool of mesh2faust². The tool takes a volumetric mesh as input and performs Finite Element Analysis (FEA) on the triangles of the mesh and generates a corresponding physical model contained in a static library file with the extension .lib. Furthermore, several additional arguments are provided to further define and improve the analysis of the mesh. Mesh2Faust is dependent on the libraries of Intel Math Kernel Library (Intel MKL)³, ARPACK⁴, and Vega FEM⁵. For this specific implementation, the FAUST repository was installed on Linux Mint⁶ and run in a Virtual Box (VB)⁷ on a PC with Windows 10 installed.

Once FAUST was installed, the tools of mesh2faust and faust2unity were used to compile the Unity plugin. Initially, the command of mesh2faust requires a 3D mesh as input. The 3D mesh of a percussive bar has previously designing and modelled by the developers of FAUST, and the specific mesh in use for the perceptual experiment can be found in the online documentation⁸. The mesh can be seen below.

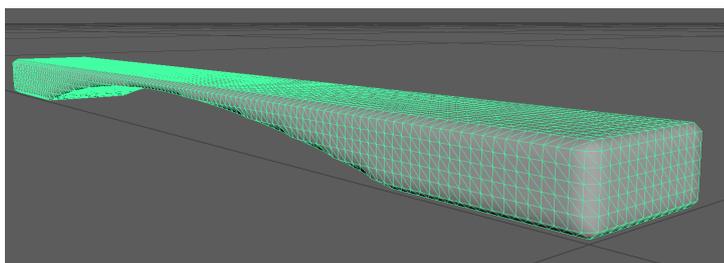


Figure 5.4: The 3D model of the bar

Mesh2Faust

As previously mentioned, mesh2faust performs FEA on the mesh of the 3D model that a physical model is intended to be generated from. The mesh should be exported as a Wavefront OBJ file (.obj), which contains data about vertex positions, normals,

²Mesh2faust: <https://github.com/grame-cncm/faust/tree/master-dev/tools/physicalModeling>

³Intel MKL: <https://software.intel.com/en-us/mkl>

⁴ARPACK: <http://www.caam.rice.edu/software/ARPACK/>

⁵Vega FEM Library: <http://run.usc.edu/vega/>

⁶Linux Mint: <https://www.linuxmint.com/>

⁷Oracle VM Virtual Box: <https://www.virtualbox.org/>

⁸FAUST Tutorials: <https://ccrma.stanford.edu/~rmichon/faustTutorials/>

faces, and UV/texturing coordinates. A physical model can be generated by navigating to the folder containing the mesh (“cd [directory]” on Linux) and running the following command in the terminal:

```
mesh2faust --infile filename.obj --name toneBar
```

The command results in a generated model with default parameters (e.g., material properties of aluminium) and the name `toneBar`. Furthermore, arguments defining frequency control, modes (min. and max. frequency), and excitation positions can be provided. The full list of arguments can be found in the aforementioned Github repository of FAUST.

For the implementation in this project, the argument of `-material` was used to define five bars of different materials. The arguments require three values related to the mechanical properties of the material; Young’s Modulus, Poisson’s Ratio, and the density. These properties were described in chapter 2 and the common materials of wood, glass, stone, plastic, and aluminium were chosen for the implementation and experiment. The mechanical material properties were obtained from Polymer Data Handbook ⁹ and the online repository of The Engineering Toolbox ¹⁰. The chosen materials are described in common terms and are generalisations due to the vast variety of types of each material. The values of the properties can be seen in the table below:

Table 5.3: My caption

Material	Young’s Modulus ($\frac{N}{m^2}$)	Poisson’s Ratio (n/a)	Density $\frac{kg}{m^3}$
Aluminium	70 e9	0.35	2700
Wood	10 e9	0.43	800
Glass	70 e9	0.23	2400
Stone	55 e9	0.25	2400
Plastic	2.4 e9	0.40	1400

Thereby, the values can be provided to the commands for compiling the physical models of the bar with various materials. The physical models were compiled with the following commands:

```
//Wood
mesh2faust --infile bar.obj --nsynthmodes 50 --nfemmodes 200 --maxmode 15000 ←
  --expos 3624 3975 4403 --debug --freqcontrol --material 10E9 0.43 800 ←
  --name woodBarModel

//Glass
mesh2faust --infile bar.obj --nsynthmodes 50 --nfemmodes 200 --maxmode 15000 ←
  --expos 3624 3975 4403 --debug --freqcontrol --material 70E9 0.23 2400 ←
  --name glassBarModel
```

⁹The Polymer Data Handbook: <https://app.knovel.com/web/toc.v/cid:kpPDHE0004>

¹⁰Engineering Toolbox: <http://www.engineeringtoolbox.com/material-properties-t24.html>

```

//Stone
mesh2faust --infile bar.obj --nsynthmodes 50 --nfemmodes 200 --maxmode 15000↔
  --expos 3624 3975 4403 --debug --freqcontrol --material 55E9 0.25 2600 ↔
  --name stoneBarModel

//Plastic
mesh2faust --infile bar.obj --nsynthmodes 50 --nfemmodes 200 --maxmode 15000↔
  --expos 3624 3975 4403 --debug --freqcontrol --material 2.4E9 0.4 1400 ↔
  --name plasticBarModel

//Aluminium
mesh2faust --infile bar.obj --nsynthmodes 50 --nfemmodes 200 --maxmode 15000↔
  --expos 3624 3975 4403 --debug --freqcontrol --material 70E9 0.35 2700 ↔
  --name aluBarModel

```

Faust2Unity

The FAUST distribution includes a collection of scripts for generating plugins for various audio platforms (e.g., C-sound, Pure Data, Max, VST, SuperCollider, etc.) from FAUST DSP programs. For this project, the Faust2Unity script was used to compile the Unity plugin from the DSP program and the library file generated from Mesh2Faust. The DSP files for each bar was set up in a similar manner as seen below:

```

import("stdfaust.lib");
import("modalModel.lib");

toneBar = modalModel(freq,exPos,t60,t60DecayRatio,t60DecaySlope)
with{
  freq = 65.4;
  exPos = 0;
  t60 = 0.1;
  t60DecayRatio = 1;
  t60DecaySlope = 5;
};

excitation = button("gate");

process = excitation : toneBar <: _,-;

```

The modalModel.lib contains the data of the FEA analysis and takes five arguments that configures the real-time sound synthesis. The GUI element of excitation was applied to a button working as a gate to enable sound output. The Faust2Unity process also generates a C# script, which is used in Unity for C# marshalling needed to control the synthesis in real-time. The Linux command for compiling the plugin for Unity 64-bit can be seen below.

```
faust2unity -w64 bar.dsp
```

The C script provides control for initialization of the synthesis parameters and can be accessed in other scripts by the number of the parameter and the SetFloatParam-

eter() function. *AudioManager* scripts were written in C# to control the behaviour of the synthesis. The behaviour was controlled using mapping of keyboard buttons to enable the gate of the output for certain models (see Appendix A for code).

Virtual Environment

In order to create a visual association with certain materials, UV-mapped textures were created to fit the 3D model of the bar. The textures were made using Blender ¹¹, GIMP ¹², and finally Unity itself. Initially, a texture image of each material (i.e., stone, wood, aluminium, glass, and plastic) were obtained from the internet. The UV-mapping coordinates were obtained using Blender, where the 3D model of the bar was imported and the unwrapping of faces were performed. The UV-map was then exported as a PNG-file and imported in GIMP for applying the texture image and remove imaging beyond the borders of the UV-coordinates. The texture image was then imported in Unity and applied to the Bumped/Diffuse Shader, which is prepacked with Unity. The shader requires a normal map, which was generated using the open-source Github repository Normal-Map Online developed by Christian Petry ¹³. When the texture image and normal map is configured in the shader options, the material can be applied to the gameobject of the bars. Multiple point lights and a reflection probe were setup in the scene to enable the material's reflection of light. The resulting visuals can be seen below.

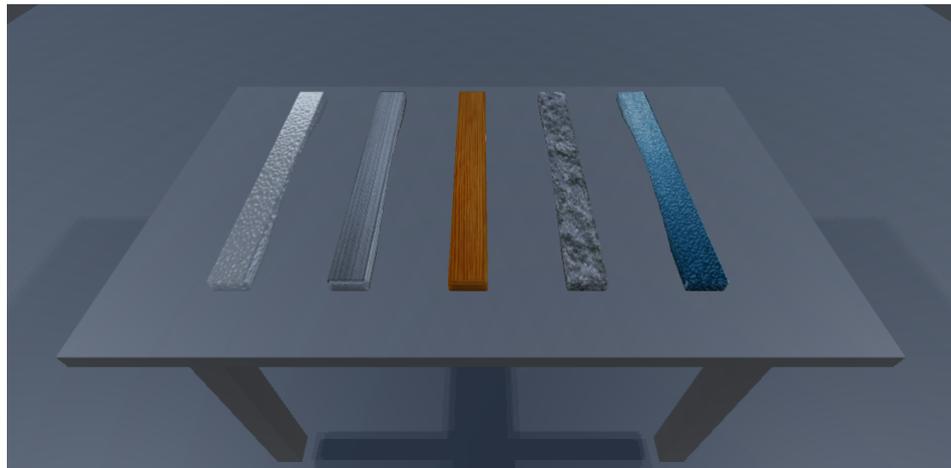


Figure 5.5: The virtual environment containing the five visual objects of various materials

¹¹Blender: <https://www.blender.org/>

¹²GIMP: <https://www.gimp.org/>

¹³Normal Map Online: <https://github.com/cpetry/NormalMap-Online>



Figure 5.6: The virtual environment containing the five visual objects of various sizes



Figure 5.7: The virtual environment containing one octave C-major scale objects for free interaction

Chapter 6

Evaluation

The following chapter will present and discuss the results of the two evaluations conducted; the usability test of the interaction prototype and the crossmodal association experiment.

6.0.1 Usability Testing

Evaluation of the first prototype was conducted on ten test participants (9M/1F, average age 25.2 years). The participants were gathered through convenience sampling [14] and were not within a defined target group, as the project's initial usability testing is not oriented towards a specific group of participants. Three out of the test participants had previous experience with musical instruments, however, none were to be considered experts in the field. All of the participants had previous experience with VR from a user perspective, and thereby, not from a technical perspective on the implementation of VR. The duration of the tests spanned within in the interval of 10 to 20 minutes, mainly due to the instruction of being free to explore the interaction for as long as a test participant would feel was needed. Test participants were mainly asked to explore and evaluate the features of spawning, translating, rotating, and scaling virtual tone bars with the physical models attached to them. After spawning and adjusting the tone bars, the test participants were able to interact and play tones by colliding with the tone bars. After each test, an interview was conducted to evaluate the experience and responses were transcribed in note-form. The questions were related to the design principles as proposed by [1], and mainly concerned the factors influencing playability (e.g., responsiveness, configuration, and limitations). Additionally, test participants were offered to contribute with comments on the overall experience, as well as feature improvement and addition. The full transcriptions can be found in the Appendix A. The main points from the qualitative evaluation can be found below in a merged form as interpreted by the author of the thesis project. The identified issues and feedback are:

- High-latency/Unregistered triggering motions
- Scaling feature was hard to operate

- Needs a frequency mapped to note option (incremental scaling)
- Needs an octave option
- “Boring” and "non-percussive sounds (provide more presets for instrument that are "drum-like")
- Looping option or beat option (looping real-time or "offline" beats/metronome)

From the qualitative data, the issues involving the playability seem to stem from the approach of real-time collision detection for striking interaction. Ideas on further improvements can be found Chapter 7.

6.0.2 Crossmodal Association

In the evaluation of the physical models ability to generate crossmodal association related to visuals, the test participants were presented with a match-to-sample task. Two groups of five varying auditory stimulus were presented to the test participants. Respectively, the groups had one variable each; size and material of the tone bar. The task of the test participants were thereby to listen to a sound, identify the associated visual object as sound producing, mark the associated object in the questionnaire and repeat for the next auditory stimulus. For each variable, this procedure was repeated five times. The frequencies related to size were implemented by increments of 100 Hz starting from 200 Hz. In regards to the properties of the material, section 5.0.5 contains a table of the values related to each material. The test was conducted on 15 test participants (11M/4F, average age 28.6 years). Test participants were found through convenience sampling [14]. The order of stimulus related to size and material of the sound producing bar was provided in a pseudo-random manner by three predefined groups. The non-randomized subgroups and stimulus types can be seen in the table below.

	S1	S2	S3	S4	S5
Size	Small	Semi-Small	Medium	Semi-Large	Large
Material	Wood	Glass	Stone	Plastic	Aluminium

Table 6.1: Table with test variables

The replies of test participants were obtained through a questionnaire provided in a Google Form. The questionnaire consisted of three parts; a preliminary section for personal information (i.e., age, gender and occupation) and multiple choice questions for the evaluation of the experiment with size and materials as variables. The questions were provided five times for each experiment and contained the following statements:

- Please choose the virtual object with the size that you associated the sound with.
- Please choose the virtual object with the material that you associated the sound with.

The statements had five options each related to the variables of size and material (see Table 6.1 for options).

Data Results

The data results' of the questionnaire has been resorted to maintain the same order of questions and stimuli. The reordered data results can be found in the table below. The table is organized by the two sub tests of *Size* and *Material* and each stimulus number.

	S1	S2	S3	S4	S5
Size	53.33 %	33.33 %	46.67 %	33.33 %	66.67 %
Material	73.33 %	46.67 %	33.33 %	20.00 %	26.67 %

Table 6.2: Table with data results on test (Percentage of correct answers for each stimuli number and type)

Size Perception The data shows a slight tendency towards identification of extremities in the perception of object size and frequency. The *small*, *medium* and *large* have the highest success rate of respectively 55.33 %, 46.46 %, and 66.67 %, whereas the intermediates of *semi-small* and *semi-large* were only correctly identified in 33.33 % of the test participants.

Material Perception The data shows a slight tendency towards identification of the *Wood* material with success rate of 73.33 % in identifying the object material of the sound producing tone bar. The lowest success rate in identification was found for the *stone* material with a success rate of only 20.00 %.

Chapter 7

Discussion

The thesis project has investigated the potential of using physical models for synthesizing real-time audio for musical interaction in game engines. The evaluation was conducted on two factors of the interaction: usability and crossmodal associations. The purpose of conducting multiple evaluations was to:

- Identify problems and improve the usability of mapping solutions between parameters of physical models and gestures
- Evaluate crossmodal correspondence with physical modelling using FEA of a 3D object

The outcome and potential reasons to problems will be discussed in the following sections.

7.0.1 Interaction Prototype (VR)

From the conducted evaluation consisting of several stages, the design and implementation of musical interaction in VR has been modified. The evaluation consisted of short usability tests throughout the development process. A mid-test was conducted on ten participants for the evaluation of the first interaction approach. The initial approach relied on real-time collision detection between a static sound generating object in 3D space and the motion captured gestures of the user. The test results showed several bugs in the implementation, which ultimately disabled the playability in a musical performance context. Through subjective analysis of the qualitatively structured interviews conducted after user testing, the conclusion was found to lie within the approach of using an unmodified version of the standard collision detection system natively supported by Unity in its PhysX physics engine.

The evaluated parameters were presented in Chapter 3 and outlined in Chapter 6. For now, the implementation has proven to be sufficient in terms of establishing a sensation of presence, comfortability through an ergonomic VR system, partly body-ownership by abstract representation of hand positions through hand-held controllers, and a minimal level of cybersickness induced by sensory mismatching. These qualities

are suspected to have been present due to the use of standardized libraries for setting up room-scale motion tracking, virtual interaction between user and environment, and high-end VR hardware (i.e., SteamVR, VRTK, and the HTC Vive). However, the influence of the remaining factors of further multimodality (i.e. addition of haptic feedback), making use of existing skills, full body-ownership, and adding a social element to the experience should be further evaluated in future work. Additional features for improving the practical use purpose of the implementation was also pointed out during evaluation. During evaluation, the test participants were given the opportunity of expressing their opinion in regards to further improvement of the entire experience. Further options for use in a scenario of musical tutoring were expressed by several test participants. They included the addition of a looping mechanism of the musical interaction, the presence of other musical instruments presented in a looping matter for matching the rhythms, more visual indications of the interaction, and the opportunity to switch between octaves in the sound synthesis of the physical models.

Improvements From the first evaluation, it was decided that a new approach should be investigated. Given the number of work-around solution that would be needed for collision detection system to work, the new approach changed the interaction from being based on impacts to more continuous motions resulting in sounds. The implementation was very preliminary and is therefore not thoroughly described in technical details in Chapter 5, however, even with the short period of development, the approach seems promising for certain types of musical instruments, which gestures already involve continuous motions. In short, the user would instead of striking sound generating rectangular boxes, submerge their hands into areas containing various physical models. During the submersion, data related to motions were calculated and mapped to parameters of the physical models.

Reiterated Design

It was found that a new approach to the interaction with the physical models had to be taken. In conjunction with the work for the crossmodal association experiment, a number of features were implemented for traditional interaction by keyboard with the physical models. The details of the revisited design can be found in Chapter 5. The new approach was needed, due to problems identified in the preliminary usability test of the first prototype. Currently, there is more work needed to be done on the behaviour and data formatting of the input provided to the physical models. In other words, the complex data from motion tracking was not sufficiently organized before mapping in the first prototype. A decision was therefore taken on taking a few steps back in terms of complexity and provide interaction with the physical modelling synthesis through keyboard input. Heuristically speaking, the new design allows for robust interaction, however, a larger scale usability (playability in regards to musical performance) study needs to be conducted in the future to verify this statement.

7.0.2 Crossmodal Experiment

An experiment on the ability of the physical models to generate crossmodal associations between the auditory and visual stimuli was conducted. The experiment tested the influence of sound frequencies on the associated visual objects, as well as the influence on visual association by parametric adjustments to the mechanical properties of the physical models generated by FEA.

Size Initially, the test results show a difficulty among test participants in identifying intermediate sizes (i.e., semi-small and semi-large) by the crossmodal associations. This meant that a higher percentage of participants were able to identify the correct size of the middle and the extremities of the range in the stimulus (i.e., small, medium and large). The difficulty with intermediate stimulus types is suspected to be related to an unfocused target group - musically trained participants could potentially be better at identifying all associations, however, initially the experiment was thought as to be perceptual exploration and therefore open to most groups of participants.

Material The second crossmodal experiment revolved around the identification of the material of the tone bar. Five different collections of material properties (wood, glass, stone, plastic, and aluminium) were used as input parameters to the process of *Mesh2Faust*, which uses FEA to generate a physical model from a 3D object. The materials were visualized in Unity using their *Material* component, which made use of diffused shading with image textures and normal maps. The test results showed that participants were able to identify a tone bar of wooden material by 73.33 %. The rest of the four materials were found to be difficult to identify with success rates of 20.00 - 46.67 %. The difficulty is suspected to be related to the input values provided to the *Mesh2Faust* process. The values of mechanical properties were found through data books and might be prone to adjustments. When looking at the values provided to *Mesh2Faust*, it can be noticed that the range of values in the command for wood differs from the rest. The high value of Poisson's ratio and low value of density might have been differentiated the auditory stimulus to such an extent that it was more easily identified than the rest of materials.

7.0.3 Future Perspective

From the findings of the prototype evaluations and crossmodal association experiments, a number of valuable objectives for future work can be drawn. The evaluations of the prototypical design showed that continuous motions for interaction with the physical models might potentially work the best with an unmodified collision detection system. Preliminarily, this issue should not be understood as impossible, but instead it can be concluded that the initial design and implementation focused too much on configurative aspects of the experience. A feasible design and implementation of collision detection for percussive interaction in VR needs to be established.

Furthermore, a larger scale usability study on the mapping between gestures and synthesis models' parameters can be conducted after establishment of feasible implementation.

In regards to the perceptual experimentation performed in the thesis project, the main problem with the conducted evaluation seems to stem from an unfocused target group. The aspect of crossmodal associations related to size might arguably be an ideal testing scenario for trained musicians, however, the initial steps show a potential inducement of realism by the definition of mechanical properties of sound producing objects.

Chapter 8

Conclusion

In this thesis, the area of physics-based sound synthesis intended for musical interaction in VR has been examined. A main objective for the thesis project was to investigate the possibilities of providing synthesized audio within a game engine to avoid the intrusiveness of previous methods for delivering dynamic audio for virtual environments. By the analysis of the fields of MR (VR in particular), research on musical interaction in VR, physical modelling synthesis, and programming and compiling options, a set of design requirements were presented in Chapter 4. From evaluation of multiple design approaches, it was concluded that the initial implementation did not deem successful in delivering real-time sound synthesis in a robust manner, which ultimately had negative influence on the playability of the VRMIs. A redesign and new approach was needed, which led to a new implementation with robustness of the sound synthesis engine in focus. By the improvements of the behaviour of input to the physical models, the design was feasible for further evaluation. The new evaluation revolved around static observation by test participants for the purpose of evaluating the sound from a perceptual point of view. Data regarding crossmodal associations generated by the auditory stimulus (i.e., sound synthesis by physical models) and visual stimulus (i.e., texturing and shading of 3D objects) was obtained through a match-to-sample task with a corresponding questionnaire. The crossmodal associations were found to be most accurate in extremities related to the size of the sound producing objects and most accurate for the physical model with the most unique mechanical properties (i.e., a wooden material). The evaluation is currently inconclusive due to the limited sample size and unfocused target group, however, it remains promising for future work as a method of evaluating the realism of physical models in use with corresponding visual stimuli.

Bibliography

- [1] S. Serafin, C. Erkut, J. Kojs, N. C. Nilsson, and R. Nordahl, “Virtual reality musical instruments: State of the art, design principles, and future directions,” *Computer Music Journal*, vol. 40, pp. 22–40, Sept 2016.
- [2] S. Serafin, A. Adjorlu, N. Nilsson, L. Thomsen, and R. Nordahl, *Considerations on the use of Virtual and Augmented Reality Technologies in Music Education*. United States: IEEE, 2017.
- [3] D. Young and S. Serafin, “Playability evaluation of a virtual bowed string instrument,” in *NIME*, 2003.
- [4] F. Miller, A. Vandome, and J. McBrewster, *Hornbostel-Sachs: Musical Instrument Classification, Erich Von Hornbostel, Curt Sachs, Musical Instrument, Ethnomusicology, Organology, Victor-Charles Mahillon, Dramaturgy, Natya Shastra*. Alphascript Publishing, 2010.
- [5] J. K. Moore, “On concepts and classifications of musical instruments,” *Musico-logy Australia*, vol. 15, no. 1, pp. 108–109, 1992.
- [6] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino, “Augmented reality: a class of displays on the reality-virtuality continuum,” vol. 2351, pp. 282–292, 1995.
- [7] C. Anthes, R. J. García-Hernández, M. Wiedemann, and D. Kranzlmüller, “State of the art of virtual reality technology,” in *2016 IEEE Aerospace Conference*, pp. 1–19, March 2016.
- [8] K. Kilteni, J.-M. Normand, M. V. Sanchez-Vives, and M. Slater, “Extending body space in immersive virtual reality: A very long arm illusion,” *PLOS ONE*, vol. 7, pp. 1–15, 07 2012.
- [9] D. Johnson and G. Tzanetakis, “Vrmin: Using mixed reality to augment the theremin for musical tutoring,” vol. 17, 2017.
- [10] R. Michon, “The faust synthesis toolkit: A set of linear and nonlinear physical models for the faust programming language,” *The Journal of the Acoustical Society of America*, vol. 134, no. 5, pp. 4054–4054, 2013.

- [11] P. R. Cook and G. P. Scavone, “The synthesis toolkit (stk,” in *Computer Music Association*, pp. 164–166, 1999.
- [12] J. O. Smith, “Physical modeling synthesis update,” *Computer Music Journal*, vol. 20, no. 2, pp. 44–56, 1996.
- [13] D. Logan, *A First Course in the Finite Element Method*. Thomson, 2007.
- [14] J. Lazar, J. H. Feng, and H. Hochheiser, *Research Methods in Human-Computer Interaction*. Wiley Publishing, 2010.
- [15] M. Schrepp, A. Hinderks, and J. Thomaschewski, *Applying the User Experience Questionnaire (UEQ) in Different Evaluation Scenarios*, pp. 383–392. Cham: Springer International Publishing, 2014.
- [16] D. A. Bowman, J. L. Gabbard, and D. Hix, “A survey of usability evaluation in virtual environments: Classification and comparison of methods,” *Presence: Teleoper. Virtual Environ.*, vol. 11, pp. 404–424, Aug. 2002.
- [17] K. S. Hale and K. M. Stanney, *Handbook of Virtual Environments: Design, Implementation, and Applications*. Boca Raton, FL, USA: CRC Press, Inc., 2nd ed., 2014.

Appendix A

Acronyms

VR	Virtual Reality
VRMI	Virtual Reality Musical Instrument
VMI	Virtual Musical Instrument
MR	Mixed Reality
AR	Augmented Reality
AV	Augmented Virtuality
NIME	New Interfaces for Musical Expression
VE	Virtual Environment
HMD	Head-Mounted Display
6DoF	6-Degrees-of-Freedom
OSC	Open Sound Control
FEA	Finite Element Analysis
FEM	Finite Element Method
CGI	Computer Generated Imagery
VRTK	Virtual Reality ToolKit
DMI	Digital Musical Instrument
FAUST	Functional Audio Stream
MIDI	Musical Instrument Digital Interface
DLL	Dynamic-Link Library
SDK	Software Development Kit
ADSR	Attack-Decay-Sustain-Release
GUI	Graphical User Interface
DAW	Digital Audio Workstation

Table A.1: Acronyms

Appendix B

Scripts

MaterialAudioManagerScript

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MaterialAudioManagerScript : MonoBehaviour {
    public FaustPlugin_aluBar aluBar;
    public FaustPlugin_woodBar woodBar;
    public FaustPlugin_glassBar glassBar;
    public FaustPlugin_plasticBar plasticBar;
    public FaustPlugin_stoneBar stoneBar;

    public int NoteOn;
    public int NoteOff;

    public GameObject aluBarObject;
    public GameObject woodBarObject;
    public GameObject glassBarObject;
    public GameObject plasticBarObject;
    public GameObject stoneBarObject;

    // Use this for initialization
    void Start () {

        aluBarObject = GameObject.FindWithTag("Alu");
        woodBarObject = GameObject.FindWithTag("Wood");
        glassBarObject = GameObject.FindWithTag("Glass");
        plasticBarObject = GameObject.FindWithTag ("Plastic");
        stoneBarObject = GameObject.FindWithTag("Stone");

        aluBar = aluBarObject.GetComponent<FaustPlugin_aluBar> ();
        woodBar = woodBarObject.GetComponent<FaustPlugin_woodBar> ();
        glassBar = glassBarObject.GetComponent<FaustPlugin_glassBar> ();
        plasticBar = plasticBarObject.GetComponent<FaustPlugin_plasticBar> ();
        stoneBar = stoneBarObject.GetComponent<FaustPlugin_stoneBar> ();
    }

    // Update is called once per frame
    void Update () {
```

```

    if (Input.GetKeyDown (KeyCode.Q)) {
        glassBar.SetFloatParameter (0, 1);
    }
    if (Input.GetKeyDown (KeyCode.W)) {
        aluBar.SetFloatParameter (0, 1);
    }
    if (Input.GetKeyDown (KeyCode.E)) {
        woodBar.SetFloatParameter (0, 1);
    }
    if (Input.GetKeyDown (KeyCode.R)) {
        stoneBar.SetFloatParameter (0, 1);
    }
    if (Input.GetKeyDown (KeyCode.T)) {
        plasticBar.SetFloatParameter (0, 1);
    }
    else if (Input.anyKey == false) {
        aluBar.SetFloatParameter (0, 0);
        woodBar.SetFloatParameter (0, 0);
        glassBar.SetFloatParameter (0, 0);
        plasticBar.SetFloatParameter (0, 0);
        stoneBar.SetFloatParameter (0, 0);
    }
}
}
}

```

SizeAudioManagerScript

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SizeAudioManagerScript : MonoBehaviour {
    public FaustPlugin_tunedBar tunedBar;
    public float frequency;
    public float initialFrequency;
    public int note;
    public int excitationSel;

    public Shader shaderNoOutline;
    public Shader shaderOutline;

    public GameObject cBar;
    public GameObject eBar;
    public GameObject gBar;
    public GameObject aBar;
    public GameObject bBar;

    // Use this for initialization
    void Start () {
        tunedBar = gameObject.GetComponent<FaustPlugin_tunedBar> ();
        initialFrequency = tunedBar.getFloatParameter (0);
        initialFrequency = frequency;
        excitationSel = 1;
        tunedBar.SetFloatParameter (8, excitationSel);

        shaderNoOutline = Shader.Find("Legacy Shaders/Bumped Diffuse");
        shaderOutline = Shader.Find("Toon/Basic Outline");

        cBar = GameObject.FindWithTag("C");
    }
}

```

```

eBar = GameObject.FindWithTag("E");
gBar = GameObject.FindWithTag("G");
aBar = GameObject.FindWithTag("A");
bBar = GameObject.FindWithTag("B");
//c, d, e, f, g, a, b
}

// Update is called once per frame
void Update () {
    if (Input.GetKeyDown(KeyCode.Q)) {
        frequency = 200.00F;
        note = 1;
    }
    if (Input.GetKeyDown(KeyCode.W)) {
        frequency = 300.00F;
        note = 1;
    }
    if (Input.GetKeyDown(KeyCode.E)) {
        frequency = 400.00F;
        note = 1;
    }
    if (Input.GetKeyDown(KeyCode.R)) {
        frequency = 500.00F;
        note = 1;
    }
    if (Input.GetKeyDown(KeyCode.T)) {
        frequency = 600.00F;
        note = 1;
    }

    } else if (Input.anyKey == false) {
        note = 0;
    }
    tunedBar.SetFloatParameter(2, note);
    tunedBar.SetFloatParameter(0, frequency);
}
}

```

AudioManagers for FAUST-STK Physical models

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TunedBarManagerScript : MonoBehaviour {
    public FaustPlugin_tunedBar tunedBar;
    public float frequency;
    public float initialFrequency;
    public int note;
    public int excitationSel;

    public Shader shaderNoOutline;
    public Shader shaderOutline;

    public GameObject cBar;
    public GameObject dBar;
    public GameObject eBar;
    public GameObject fBar;
}

```

```

public GameObject gBar;
public GameObject aBar;
public GameObject bBar;
public GameObject c2Bar;

// Use this for initialization
void Start () {
    tunedBar = gameObject.GetComponent<FaustPlugin_tunedBar> ();
    initialFrequency = tunedBar.getFloatParameter (0);
    initialFrequency = frequency;
    excitationSel = 1;
    tunedBar.SetFloatParameter (8, excitationSel);

    shaderNoOutline = Shader.Find("Legacy Shaders/Bumped Diffuse");
    shaderOutline = Shader.Find("Toon/Basic Outline");

    cBar = GameObject.FindWithTag("C");
    dBar = GameObject.FindWithTag("D");
    eBar = GameObject.FindWithTag("E");
    fBar = GameObject.FindWithTag("F");
    gBar = GameObject.FindWithTag("G");
    aBar = GameObject.FindWithTag("A");
    bBar = GameObject.FindWithTag("B");
    c2Bar = GameObject.FindWithTag("C2");
    //c, d, e, f, g, a, b
}

// Update is called once per frame
void Update () {

    if (Input.GetKeyDown (KeyCode.Q)) {
        frequency = 261.63F;
        note = 1;
        cBar.GetComponent<Renderer> ().material.shader = shaderOutline;
    }
    if (Input.GetKeyDown (KeyCode.W)) {
        frequency = 293.66F;
        note = 1;
        dBar.GetComponent<Renderer> ().material.shader = shaderOutline;
    }
    if (Input.GetKeyDown (KeyCode.E)) {
        frequency = 329.63F;
        note = 1;
        eBar.GetComponent<Renderer> ().material.shader = shaderOutline;
    }
    if (Input.GetKeyDown (KeyCode.R)) {
        frequency = 349.23F;
        note = 1;
        fBar.GetComponent<Renderer> ().material.shader = shaderOutline;
    }
    if (Input.GetKeyDown (KeyCode.T)) {
        frequency = 392.00F;
        note = 1;
        gBar.GetComponent<Renderer> ().material.shader = shaderOutline;
    }
    if (Input.GetKeyDown (KeyCode.Y)) {
        frequency = 440.00F;
        note = 1;
        aBar.GetComponent<Renderer> ().material.shader = shaderOutline;
    }
    if (Input.GetKeyDown (KeyCode.U)) {

```

```
frequency = 493.88F;
note = 1;
bBar.GetComponent<Renderer> ().material.shader = shaderOutline;
}
if (Input.GetKeyDown (KeyCode.I)) {
frequency = 523.25F;
note = 1;
c2Bar.GetComponent<Renderer> ().material.shader = shaderOutline;
} else if (Input.anyKey == false) {
note = 0;
cBar.GetComponent<Renderer> ().material.shader = shaderNoOutline;
dBar.GetComponent<Renderer> ().material.shader = shaderNoOutline;
eBar.GetComponent<Renderer> ().material.shader = shaderNoOutline;
fBar.GetComponent<Renderer> ().material.shader = shaderNoOutline;
gBar.GetComponent<Renderer> ().material.shader = shaderNoOutline;
aBar.GetComponent<Renderer> ().material.shader = shaderNoOutline;
bBar.GetComponent<Renderer> ().material.shader = shaderNoOutline;
c2Bar.GetComponent<Renderer> ().material.shader = shaderNoOutline;
}

tunedBar.SetFloatParameter (2, note);
tunedBar.SetFloatParameter (0, frequency);
}
}
```


Appendix C

Virtual Reality Technology

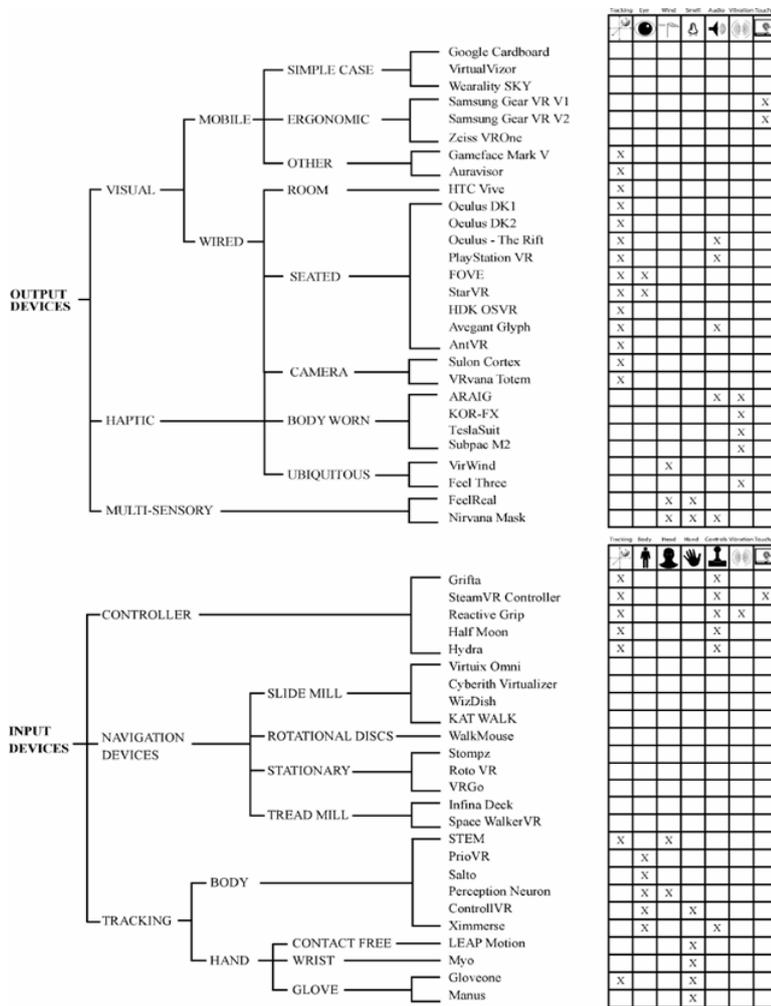


Figure C.1: State of the Art in Virtual Reality Technology [7]

Appendix D

Musical VR Applications

Music Room

The Music Room ¹ is a collection of instruments designed for the HTC Vive. By using the hand-held controllers of the HTC Vive, the platform enables interaction with a set of percussive instruments – both pitched and unpitched. Moreover, the philosophy around The Music Room is that it acts as a MIDI controller for use in any DAW. The software package includes the DAW Bitwig for instant integration. Additionally, The Music Room also supports MIDI triggers for example adding of a kick drum operated by the foot – a fourth tracking point that the HTC Vive currently does not support, apart from the tracking of head and hand orientation and position. However, with the recent introduction of the HTC Vive Tracker, the need for additional equipment outside the HTC Vive system might not be needed.

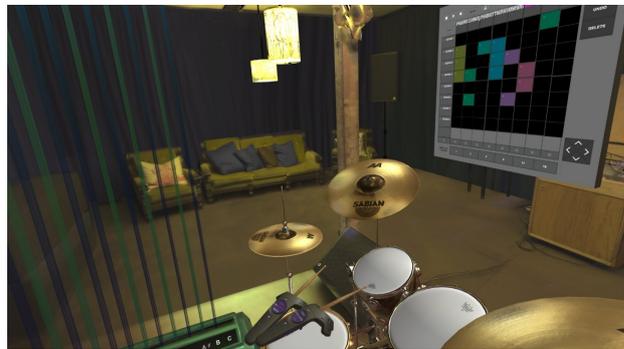


Figure D.1: Music Room

¹Music Room: <http://musicroomvr.com/>

Soundstage VR

Soundstage VR ² is a music instrument sandbox. Similar to The Music Room, the Soundstage VR is also compatible with the HTC Vive with its room-scale motion tracking. Apart from the interactive VRMIs, the sandbox also includes a modular mix chain with a library of effects and processing. Soundstage VR also implements a looping and recording stage for use in post-production or other media productions.



Figure D.2: Soundstage VR

Percussive VR

Percussive VR ³ consists of a collection of VRMIs such as replications of a marimba, steel drums, and glockenspiel. The platform is still in the early development stages; however, the aim of the experience seems to put more weight on the practice of melodic and rhythmic performances.

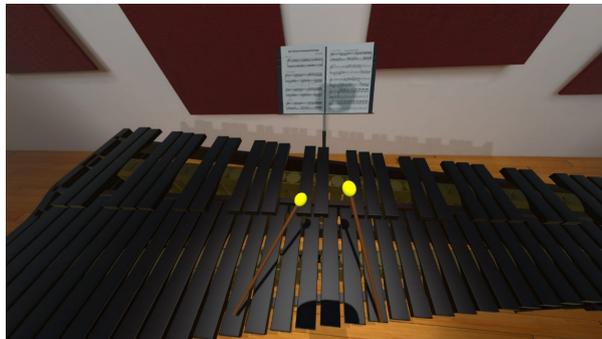


Figure D.3: Percussive VR

²SoundstageVR: <http://www.soundstagevr.com/>

³Percussive VR: <http://store.steampowered.com/app/536370/PercussiveVR/>

LyraVR

LyraVR ⁴ is a fully configurable musical platform for composition of and interaction with musical sequences. The platform features GUI interfaces for configuration and playback behaviour, while being designed with a node approach for connecting musical instruments.



Figure D.4: LyraVR

Stage Presence

A slightly different approach to a musical experience in VR is that of the yet to be released game Stage Presence ⁵. The game is operated using a person's voice to sing with a band on stage in front of a virtual crowd. There can be said to be an absence of VRMIs as compared to the solutions presented, however, still incorporates musical expressions by the user's singing voice as instrument.



Figure D.5: Stage Presence

Music Inside: A VR Rhythm Game

Currently compatible with both the HTC Vive and Oculus Rift (incl. Oculus Touch for interaction), the game Music Inside ⁶ has similar elements to the classic digital

⁴LyraVR: <http://lyravr.com/>

⁵Stage Presence: <http://store.steampowered.com/app/391640/Stagepresence/>

⁶Music Inside: <http://www.musicinsidevr.com/>

musical game of Guitar Hero. The user is presented with a percussive setup and receives visual cues related to a rhythmic task that is motivated to the user by the scoring of points related to performance.



Figure D.6: Music Inside: A VR Rhythm Game

AudioShield

AudioShield ⁷ is a rhythmic game that takes any song (or streamed song from a database) as parametric input to the game play. The aim is for the user to maintain rhythm related to the provided song by hitting a stream of visual cues synchronized with the music. AudioShield is currently compatible with the HTC Vive system.

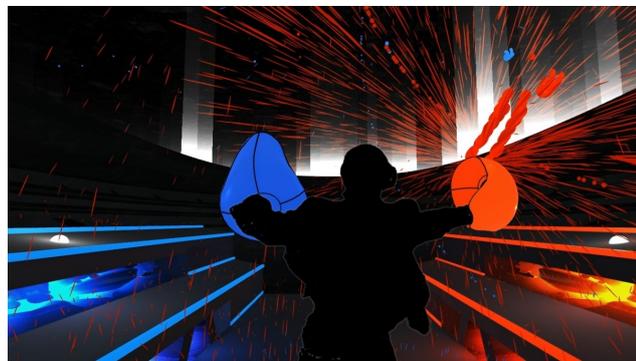


Figure D.7: AudioShield

Holodance

Similar to AudioShield, Holodance ⁸ is a rhythmic game, however, compatible with both the HTC Vive and the Oculus Rift. Different from AudioShield, Holodance

⁷AudioShield: <http://audio-shield.com/>

⁸Holodance: <http://holodance-vr.com/>

takes input from so-called beatmaps generated with the standalone rhythmic game *osu!*. In short, the game provides the user with elements cued to the beatmaps, which the user has to interact with to advance in the game.

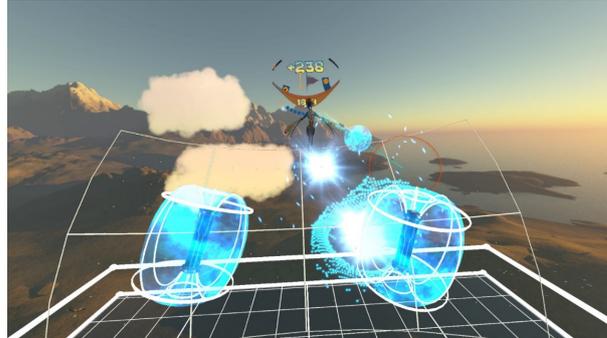


Figure D.8: Holodance

Björk VR Music Videos

In regards to purely visual experiences, that are only interactive by changing head orientation, Icelandic singer and songwriter Björk is working on an album with videos for VR. The virtual environments are made of CGI and artistically expresses the singer's vision with the songs. Two teasers for the singles of *Family VR* and *NotGet* has been released on Youtube.

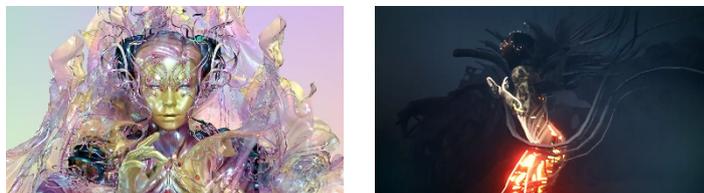


Table D.1: Music Videos in VR by Björk

Appendix E

Physical Model Parameters

- Basic Parameters
 - Frequency (Hz)
 - Gain (0-1)
 - NoteOff/NoteOn (0-1)
- Physical Parameters
 - Bow Position (0-1)
 - Bow Pressure (0-1)
 - Excitation Selector (0=Bow, 1=Strike)
 - Integration Constant (0-1)
 - Base Gain (0-1)
 - Bow Pressure (0-1)
 - Bow Position (0-1)
 - Resonance (0-1)
- Nonlinear Filter Parameters
 - Modulation Type (Incoming signal, average incoming signal, squared incoming signal, sine wave at frequency X)
 - Nonlinearity (0-1)
 - Frequency Modulation (Frequency of the modulating signal)
 - Nonlinearity Attack (Attack duration of the nonlinearity)
- Envelopes and Vibrato
 - Vibrato Frequency (Hz)
 - Vibrato Gain (0-1)

- Vibrato Begin (Vibrato silence duration before attack)
- Vibrato Attack (Vibrato attack duration)
- Vibrato Release (Vibrato release duration)
- Envelope Attack (Envelope attack duration)
- Envelope Decay (Envelope decay duration)
- Envelope Release (Envelope release duration)